12-06-99

A

JC545 U.S. PTO 12/03/99

Please type a plus sign (+) inside this box → [+]

PTO/SB/05 (4/98)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 C.F.R. § 1.53(b))

Attorney Docket No.	NAP 042
First Inventor or Application Identifier	Gaurav Banga
Title	Computer Assisted Automatic Error Detection and ...
Express Mail Label No.	EK104846394US

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231

JC542 U.S. PTO 09/456027 12/03/99

1. [ ] * Fee Transmittal Form *(e.g., PTO/SB/17)* *(Submit an original and a duplicate for fee processing)*
2. [X] Specification [Total Pages 28] *(preferred arrangement set forth below)*
   - Descriptive title of the Invention
   - Cross References to Related Applications
   - Statement Regarding Fed sponsored R & D
   - Reference to Microfiche Appendix
   - Background of the Invention
   - Brief Summary of the Invention
   - Brief Description of the Drawings *(if filed)*
   - Detailed Description
   - Claim(s)
   - Abstract of the Disclosure
3. [X] Drawing(s) *(35 U.S.C. 113)* [Total Sheets 5]
4. Oath or Declaration [Total Pages]
   - a. [ ] Newly executed (original or copy)
   - b. [ ] Copy from a prior application (37 C.F.R. § 1.63(d)) *(for continuation/divisional with Box 16 completed)*
     - i. [ ] DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 C F.R. §§ 1.63(d)(2) and 1.33(b).

NOTE FOR ITEMS 1 & 13: IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).

5. [ ] Microfiche Computer Program *(Appendix)*
6. Nucleotide and/or Amino Acid Sequence Submission *(if applicable, all necessary)*
   - a. [ ] Computer Readable Copy
   - b. [ ] Paper Copy (identical to computer copy)
   - c. [ ] Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

7. [ ] Assignment Papers (cover sheet & document(s))
8. [ ] 37 C.F.R.§3.73(b) Statement *(when there is an assignee)* [ ] Power of Attorney
9. [ ] English Translation Document *(if applicable)*
10. [ ] Information Disclosure Statement (IDS)/PTO-1449 [ ] Copies of IDS Citations
11. [ ] Preliminary Amendment
12. [X] Return Receipt Postcard (MPEP 503) *(Should be specifically itemized)*
13. [ ] * Small Entity Statement(s) *(PTO/SB/09-12)* [ ] Statement filed in prior application, Status still proper and desired
14. [ ] Certified Copy of Priority Document(s) *(if foreign priority is claimed)*
15. [X] Other: 1. Technical Appendix (16 pages, including cover sheet) 2. Certificate of Mailing

16. **If a CONTINUING APPLICATION,** *check appropriate box, and supply the requisite information below and in a preliminary amendment:*

[ ] Continuation [ ] Divisional [ ] Continuation-in-part (CIP) of prior application No: _____/_____

Prior application information: Examiner _____ *Group / Art Unit:* _____

For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

17. CORRESPONDENCE ADDRESS

[X] *Customer Number or Bar Code Label* (Insert Customer No. or Attach bar code label here) 22883 *or* [ ] *Correspondence address below*

Name	PATENT TRADEMARK OFFICE
Address	
City	State / Zip Code
Country	Telephone / Fax

Name (Print/Type)	Steven A. Swernofsky	Registration No. (Attorney/Agent)	33,040
Signature		Date	December 3, 1999

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

NAP 042

Certificate of Mailing (37 C.F.R. § 1.10)

I hereby certify that this paper (along with any paper referred to as being attached or enclosed) is being deposited with the United States Postal Services on the date shown below as "Express Mail" (Post Office to Addressee) in an envelope addressed to the Honorable Assistant Commissioner for Patents and Trademarks, Box Patent Application, Washington, D.C. 20231.

Mailing Label No. EK104846394US

Date of Deposit: December 3, 1999

John Fleming
Printed Name

John Fleming
Signature

Documents enclosed:

1. Utility Patent Application Transmittal (PTO/SB/05)
2. Application, including the following:
   - 23 pages of Description
   - 4 pages of Claims
   - 1 page Abstract of the Disclosure
   - 5 pages of drawings
3. Technical Appendix, including the following:
   - 1 page cover sheet
   - 15 pages of body of Appendix
4. Certificate of Mailing
5. Return Postcard

This application is submitted in the name of the following inventor(s):

Inventor	Citizenship	Residence City and State
Banga, Gaurav	India	San Carlos, California

The assignee is Network Appliance, Inc., a corporation having an office at 495 East Java Drive, Sunnyvale California 94089.

Title of the Invention

Computer Assisted Automatic Error Detection and Diagnosis of File Servers

Background of the Invention

1. Field of the Invention

This invention relates to computer assisted automatic error detection and diagnosis of file servers, such as for a networked file server.

2. *Related Art*

Network file servers are subject to errors and other failures (such as less-than-expected levels of performance), including those arising from hardware failure, software error, erroneous configuration, or mismatch between configuration and usage. Many of these errors and other failures are similar to those that are common for general-purpose computer systems, and these would therefore be subject to similar forms of error detection and diagnosis by a user of the system. However, many network file servers are designed for minimal user intervention, and are specifically intended to have extremely simple user interfaces. One such group of easy-to-use network file servers are called "Network Appliance"™ file servers, for which the model of use is that the device is as easy to use as a common toaster.

One problem with known systems is that relative ease of use is often coupled with relatively unsophisticated users. Error detection and diagnosis by relatively unsophisticated users is subject to the drawback that relatively unsophisticated users are relatively unsophisticated in the use of error detection and diagnosis techniques. Thus, forms of error detection and diagnosis that are acceptable for general-purpose computer systems are not nearly as suitable for network file servers where one of the most important purposes is to be easy-to-use.

Accordingly, it would be advantageous to provide a technique for computer assisted automatic error detection and diagnosis of file servers that is not subject to drawbacks of the known art.

Summary of the Invention

The invention provides a method and system for computer assisted automatic error detection and diagnosis of file servers. In a preferred embodiment, the file server includes diagnostic software modules for (1) periodic and continuous interpretation of monitoring statistics, (2) augmentation of known network protocols, (3) cross-layer analysis of monitoring statistics, and (4) tracking of hardware and software configuration changes.

In a first aspect of the invention, the diagnostic software modules periodically and continuously review monitoring statistics gathered by the file server regarding its operation. These monitoring statistics can include a wide variety of values gathered by software modules at disparate levels within the operating system of the file server. The collection of monitoring statistics is processed by a pattern matching system and a rule-based inference system. In a preferred embodiment, the pattern matching system and rule-based inference system are responsive to a use profile for the file server, such as whether the file server is being used for an ISP, a development environment, a mail server, or otherwise.

In a second aspect of the invention, the diagnostic software modules are capable of augmenting known network protocols, by manipulating parameters of lower-level protocols using different higher-level protocols. The diagnostic software modules can manipulate known parameters of the lower-level protocols in rapid succession, so as to try a large number of combinations of protocol parameters. Using the higher-level protocols, the diagnostic software modules can determine if the selected parameters for the lower-level protocols are correct.

In a third aspect of the invention, the diagnostic software modules are capable of imposing sequential and combined constraints on diagnosis of possible errors, with reference to known logical coupling between monitoring statistics gathered at multiple logical levels of software modules within the file server. In a preferred embodiment, constraints from multiple logical levels are chained together so as to limit the number of possible errors deduced as possible from the various monitoring statistics to a relatively small number.

In a fourth aspect of the invention, the diagnostic software modules are capable of tracking hardware and software configuration changes to the file server, and relating changes in known monitoring statistics to timing of those hardware and software configuration changes. In a preferred embodiment, the diagnostic software modules are capable of determining the configuration change most likely to be responsible for a com-

puter assisted diagnosed error, and of suggesting activities to reverse the hardware and software configuration changes so as to place the file server in an operating state.

The invention provides an enabling technology for a wide variety of applications for computer assisted automatic error detection and diagnosis of file servers, so as to obtain substantial advantages and capabilities that are novel and non-obvious in view of the known art. Examples described below primarily relate to networked file servers, but the invention is broadly applicable to many different types of automated software systems.

Brief Description of the Drawings

Figure 1 shows a block diagram of a system for computer assisted automatic error detection and diagnosis of file servers.

Figure 2 shows a process flow diagram of a first method for operating a system for computer assisted automatic error detection and diagnosis of file servers, including periodic and continuous interpretation of monitoring statistics.

Figure 3 shows a process flow diagram of a second method for operating a system for computer assisted automatic error detection and diagnosis of file servers, including augmentation of known network protocols.

Figure 4 shows a process flow diagram of a third method for operating a system for computer assisted automatic error detection and diagnosis of file servers, including cross-layer analysis of monitoring statistics.

Figure 5 shows a process flow diagram of a fourth method for operating a system for computer assisted automatic error detection and diagnosis of file servers, including tracking of hardware and software configuration changes.

Detailed Description of the Preferred Embodiment

In the following description, a preferred embodiment of the invention is described with regard to preferred process steps and data structures. Embodiments of the invention can be implemented using general-purpose processors or special purpose processors operating under program control, or other circuits, adapted to particular process steps and data structures described herein. Implementation of the process steps and data structures described herein would not require undue experimentation or further invention.

Lexicography

The following terms refer or relate to aspects of the invention as described below. The descriptions of general meanings of these terms are not intended to be limiting, only illustrative.

- **client** and **server** — in general, these terms refer to a relationship between two devices, particularly to their relationship as client and server, not necessarily to any particular physical devices.

  For example, but without limitation, a particular client device in a first relationship with a first server device, can serve as a server device in a second relationship with a second client device. In a preferred embodiment, there are generally a relatively small number of server devices servicing a relatively larger number of client devices.

- **client device** and **server device** — in general, these terms refer to devices taking on the role of a client device or a server device in a client-server relationship (such as an HTTP web client and web server). There is no particular requirement that any client devices or server devices must be individual physical devices. They can each be a single device, a set of cooperating devices, a portion of a device, or some combination thereof.

For example, but without limitation, the client device and the server device in a client-server relation can actually be the same physical device, with a first set of software elements serving to perform client functions and a second set of software elements serving to perform server functions

- **configuration changes** — in general, information regarding changes to a configuration of the file server

- **cross-layer analysis** — in general, a technique for applying a large number of combinations of diagnostic constraints, so as to determine a set of errors or other faults are not compatible with the set of current monitoring statistics

- **diagnostic constraint** — in general, a functional or logical constraint on possible errors or other failures, due to functional or logical structure of the file server or its operating system

- **diagnostic software module** — in general, a software module in the file server used for performing computer assisted automatic error detection and diagnosis

- **error detection and diagnosis** — in general, a technique for detecting errors and other failures, and for determining a likely cause thereof

- **lower-level** and **higher-level** protocols — in general, these terms refer to a relationship between two protocols, particularly to their relationship as a higher-level protocol which relies on operation of a lower-level protocol and which is able to alter parameters of the lower-level protocol, not necessarily to any particular protocols.

- **manipulating parameters** — in general, a technique for using a higher-level protocol to determine whether a lower-level protocol is operating relatively efficiently using a set of selected parameters for the lower-level protocol, and using the lower-level protocol to repeatedly and rapidly alter those selected parameters so as to find an optimal set of selected parameters

- **monitoring statistics** — in general, information regarding performance of the file server or other device

- **network protocol** — in general, a technique for communication between devices, such as for example between (a) the file server or other device and (b) a point external to the file server or other device.

- **pattern matching** — in general, a technique for comparing a set of monitoring statistics against a selected pattern known to be related to an error or other fault

- **periodic and continuous interpretation** — in general, repeated and rapid review of monitoring statistics, so as to identify errors or other faults rapidly and as early as possible

- **protocol augmentation** — in general, a technique for using a higher-level protocol to determine whether a lower-level protocol is operating relatively efficiently using a set of selected parameters for the lower-level protocol, and using the lower-level protocol to repeatedly and rapidly alter those selected parameters so as to find an optimal set of selected parameters

- **rule-based inference system** — in general, a technique for drawing factual conclusions from monitoring statistics and other known facts, so as to determine the presence or absence of an error or other fault

- **sequential and combined constraints on diagnosis** — in general, a technique for applying a large number of combinations of diagnostic constraints, so as to determine a set of errors or other faults are not compatible with the set of current monitoring statistics

- **usage profile** — in general, information regarding a pattern or profile of use of a file server or other device

As noted above, these descriptions of general meanings of these terms are not intended to be limiting, only illustrative. Other and further applications of the invention, including extensions of these terms and concepts, would be clear to those of ordinary skill in the art after perusing this application. These other and further applications are part of the scope and spirit of the invention, and would be clear to those of ordinary skill in the art, without further invention or undue experimentation.

System Elements

Figure 1 shows a block diagram of a system for computer assisted automatic error detection and diagnosis of file servers.

A system 100 includes a file server (or other device) 110, a communication network 120, a network interface 130, and a database 140.

The file server (or other device) 110 includes a computer having a processor, program and data memory, mass storage, a presentation element, and an input element, and coupled to the communication network 120. As used herein, the term "computer" is intended in its broadest sense, and includes any device having a programmable processor or otherwise falling within the generalized Turing machine paradigm. The mass storage can include any device for storing relatively large amounts of information,

such as magnetic disks or tapes, optical devices, magneto-optical devices, or other types of mass storage.

The file server 110 includes operating system software 111, including an appliance operating system 112 for controlling the file server 110 and performing its file server operations, and a diagnostic software module 113 for performing computer assisted automatic error detection and diagnosis.

The diagnostic software module 113 includes software modules for error detection and diagnosis (further described with regard to figure 2, figure 3, figure 4, and figure 5), and is coupled to the database 140 for storing and retrieving statistical and other information.

The communication network 120 includes any technique for sending information between the file server 110 and at least one point outside the file server 110. In a preferred embodiment, the communication network 120 includes a computer network, such as an Internet, intranet, extranet, or a virtual private network. In alternative embodiments, the communication network 120 can include a direct communication line, a switched network such as a telephone network, or some combination thereof.

The network interface 130 includes a communication link between the file server 110 and the communication network 120.

The database 140 includes memory or mass storage in which monitoring statistics and other information about the file server 110 can be recorded and retrieved therefrom.

Method of Operation (Interpretation of Monitoring Statistics)

Figure 2 shows a process flow diagram of a first method for operating a system for computer assisted automatic error detection and diagnosis of file servers, including periodic and continuous interpretation of monitoring statistics.

A method 200 includes a set of flow points and a set of steps. The system 100 performs the method 200. Although the method 200 is described serially, the steps of the method 200 can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method 200 be performed in the same order in which this description lists the steps, except where so indicated.

At a flow point 210, the diagnostic software module 113 is ready to passively monitor operating system statistics.

At a step 211, the diagnostic software module 113 receives operating system statistics from the database 140 and compares those operating system statistics against a set of predefined rules regarding statistical aberrations. If the operating system statistics fall within the predefined rules, thus indicating an abnormal system state, the diagnostic software module 113 flags that abnormal system state, enters a reporting record in the database 140, and reports the detected abnormal system state and its associated operating system statistics to an operator (not shown).

In a preferred embodiment, the predefined rules regarding statistical aberrations include both (a) pattern matching, in which the diagnostic software module 113 compares the monitoring statistics against selected patterns known to be related to an errors or other faults, and (b) rule-based inference, in which the diagnostic software module 113 draws factual conclusions from monitoring statistics and other known facts, so as to determine the presence or absence of errors or other faults.

In a preferred embodiment, the predefined rules regarding statistical aberrations are responsive to a usage pattern for the file server, as recorded in the database 140 or otherwise. For example, the file server can be primarily used as for an ISP, a development environment, a mail server, or otherwise. The usage pattern for the file server could reflect such information, preferably including an expected number of distinct users, an expected number of distinct files, an expected size for typical files, and an expected duration of existence for distinct files or file-usage sessions.

At a step 212, the diagnostic software module 113 determines, using the predefined rules, whether the abnormal system state corresponds to a specific known problem, or is associated with more than one known problem. If the abnormal system state corresponds to a specific known problem, the diagnostic software module 113 so indicates and reports the specific known problem to the operator.

At a step 213, if the abnormal system state was determined in the previous step to correspond to a set of more than one known problem, the diagnostic software module 113 determines if there is a predefined technique for analyzing the abnormal system state. If there is such a predefined technique, the diagnostic software module 113 proceeds with the next step.

At a step 214, the diagnostic software module 113 performs the predefined technique for analyzing the abnormal system state found in the previous step.

After this step, the method 200 has performed one round of passively monitoring operating system statistics, and is ready to perform another such round so as to continuously monitor operating system statistics.

The method 200 is performed one or more times starting from the flow point 210 and continuing therefrom. In a preferred embodiment, the diagnostic software

module 113 repeatedly performs the method 200, starting from the flow point 210 and continuing therefrom, so as to review monitoring statistics gathered by the file server periodically and continuously.

Method of Operation (Augmentation of Network Protocols)

Figure 3 shows a process flow diagram of a second method for operating a system for computer assisted automatic error detection and diagnosis of file servers, including augmentation of known network protocols.

A method 300 includes a set of flow points and a set of steps. The system 100 performs the method 300. Although the method 300 is described serially, the steps of the method 300 can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method 300 be performed in the same order in which this description lists the steps, except where so indicated.

At a flow point 310, the diagnostic software module 113 is ready to determine the presence or absence of errors or other faults, using protocol augmentation of known network protocols.

At a step 311, the diagnostic software module 113 selects parameters for known network protocols, so as to communicate with a device at a point outside the file server 110. In selecting these parameters, the diagnostic software module 113 attempts to choose parameters that match with the device outside the file server 110, so as to achieve relatively efficient communication.

At a step 312, the diagnostic software module 113 uses the lower-level protocol to set its parameters to the parameters selected in the previous step.

At a step 313, the diagnostic software module 113 communicates with the device outside the file server 110 using a higher-level protocol, and uses the higher-level protocol to measure the relative efficiency of communication therewith.

At a step 314, the diagnostic software module 113 selects a new set of parameters for the lower-level protocol, in response to the measure of relative efficiency determined using the higher-level protocol in the previous step.

The diagnostic software module 113 performs the step 312, the step 313, and the step 314, repeatedly and rapidly, so as to try a large number of combinations of protocol parameters. The diagnostic software module 113 thus determines which selected parameters are optimal.

At a step 315, the diagnostic software module 113 uses the lower-level protocol to set its parameters to the parameters determined to be optimal.

After this step, the method 300 has set the optimal parameters for the lower-level protocol.

The method 300 can be performed one or more times starting from the flow point 310 and continuing therefrom.

Method of Operation (Cross-Layer Analysis)

Figure 4 shows a process flow diagram of a third method for operating a system for computer assisted automatic error detection and diagnosis of file servers, including cross-layer analysis of monitoring statistics.

A method 400 includes a set of flow points and a set of steps. The system 100 performs the method 400. Although the method 400 is described serially, the steps of the method 400 can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method 400 be performed in the same order in which this description lists the steps, except where so indicated.

At a flow point 410, the diagnostic software module 113 is ready to perform cross-layer analysis of monitoring statistics.

At a step 411, the diagnostic software module 113 identifies a diagnostic constraint applicable to the set of current monitoring statistics.

At a step 412, the diagnostic software module 113 performs pattern matching and rule-based inference to attempt to identify a known error or other fault, similar to the method 200.

The diagnostic software module 113 performs the step 411 and the step 412 repeatedly and rapidly, so as to apply a large number of combinations of diagnostic constraints. The diagnostic software module 113 thus determines a set of errors or other faults are not compatible with the set of current monitoring statistics.

In a preferred embodiment, the diagnostic software module 113 continues to apply combinations of diagnostic constraints until the set of errors or other faults is reduced to only a few possibilities.

After this step, the method 400 has used cross-layer analysis of monitoring statistics to restrict the set of possible errors or other faults.

The method 400 can be performed one or more times starting from the flow point 410 and continuing therefrom.

Method of Operation (Tracking Configuration Changes)

Figure 5 shows a process flow diagram of a fourth method for operating a system for computer assisted automatic error detection and diagnosis of file servers, including tracking of hardware and software configuration changes.

A method 500 includes a set of flow points and a set of steps. The system 100 performs the method 500. Although the method 500 is described serially, the steps of the method 500 can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method 500 be performed in the same order in which this description lists the steps, except where so indicated.

At a flow point 510, the diagnostic software module 113 is ready to perform tracking of hardware and software configuration changes.

At a step 511, the diagnostic software module 113 identifies a past hardware or software configuration for the file server, recorded in the database 140 or other-

wise, and a present hardware or software configuration for the file server, recorded in the database 140 or otherwise.

At a step 512, the diagnostic software module 113 identifies a set of differences between the past configuration and the present configuration.

At a step 513, the diagnostic software module 113 identifies a set of possible errors or other faults associated with the set of differences identified in the previous step.

At a step 514, the diagnostic software module 113 applies diagnostic constraints from the set of current monitoring statistics to the set of possible errors or other faults, so as to determine which errors or other faults are most likely.

At a step 515, the diagnostic software module 113 uses the determination, in the previous step, of which errors or other faults are most likely, to suggest activities to reverse the hardware or software configuration changes so as to place the file server in an operating state.

After this step, the method 500 has used tracking of hardware and software configuration changes to restrict the set of possible errors or other faults.

The method 500 can be performed one or more times starting from the flow point 510 and continuing therefrom.

Generality of the Invention

The invention has general applicability to various fields of use, not necessarily related to the services described above. For example, these fields of use can include other "appliance"-type devices other than file servers.

Other and further applications of the invention in its most general form, will be clear to those skilled in the art after perusal of this application, and are within the scope and spirit of the invention.

Technical Appendix

The technical appendix enclosed with this application is hereby incorporated by reference as if fully set forth herein, and forms a part of the disclosure of the invention and its preferred embodiments.

Alternative Embodiments

Although preferred embodiments are disclosed herein, many variations are possible which remain within the concept, scope, and spirit of the invention, and these variations would become clear to those skilled in the art after perusal of this application.

Claims

1. A method, including steps of

repeatedly reviewing monitoring statistics regarding operation of a file server, said steps of reviewing being performed at least as often as a selected time period;

processing said monitoring statistics using a diagnostic software module, in response to said steps of repeatedly reviewing;

whereby a result of said steps of processing includes a diagnosis of a behavior of said file server.

2. A method as in claim 1, wherein said diagnostic software module includes a pattern matching system and a rule-based inference system

3. A method as in claim 1, wherein said monitoring statistics include information gathered by at least a first and at least a second software module, said first and second software modules being disposed at differing levels within an operating system of said file server.

4. A method as in claim 1, wherein said monitoring statistics include information gathered by at least one software module within an operating system of said file server.

5. A method as in claim 1, wherein said selected time period is less than 10 seconds.

6. A method as in claim 1, wherein said steps of processing are responsive to a usage profile for said file server.

7. A method as in claim 6, wherein said usage profile includes information regarding whether use of said file server includes usage as an ISP, a development environment, a mail server, or otherwise.

8. A method, including steps of

selecting a set of parameters for a first communication protocol;

attempting to communicate, between a point inside a file server and a point outside said file server, using a second communication protocol, said second communication protocol making use of said first communication protocol;

reviewing a result of said steps of attempting to communicate; and

altering said set of parameters, in response to a result of said steps of reviewing.

9. A method as in claim 8, wherein said steps of altering are performed at least as often as a selected time period of less than ten seconds.

10. A method as in claim 8, wherein said steps of altering are performed repeatedly, whereby a resulting set of parameters allows substantial communication between said first point and said second point.

11. A method as in claim 8, wherein said steps of attempting to communicate are performed using at least one hundred differing said sets of parameters.

12. A method, including steps of

imposing combined constraints on diagnosis of possible errors, in response to known logical coupling between monitoring statistics gathered at multiple logical levels of software modules within a file server; and

chaining constraints from multiple logical levels together;

whereby a number of possible errors deduced as possible from the various monitoring statistics are limited to a relatively small number.

13. A method, including steps of

tracking configuration changes to a file server;

relating changes in known monitoring statistics to timing of said hardware and software configuration changes; and

determining, in response to said steps of tracking and of relating, a configuration change most likely to be responsible for an error or other failure in said file server.

14. A method as in claim 13, including steps of suggesting activities to reverse said configuration changes so as to place said file server in an operating state.

15. A method as in claim 13, wherein said configuration changes include hardware and software configuration changes.

Abstract of the Disclosure

The invention provides a method and system for computer assisted automatic error detection and diagnosis of file servers. Software modules periodically and continuously review monitoring statistics gathered by the file server regarding its operation. The monitoring statistics are processed by a pattern matching system and a rule-based inference system. Software modules augment known network protocols, by manipulating parameters of lower-level protocols using different higher-level protocols. Software modules manipulate known parameters of the lower-level protocols in rapid succession, so as to try a large number of combinations of protocol parameters. Using the higher-level protocols, software modules determine if the selected parameters for the lower-level protocols are correct. Software modules impose sequential and combined constraints on diagnosis of possible errors, with reference to known logical coupling between monitoring statistics gathered at multiple logical levels of software modules within the file server. Constraints from multiple logical levels are chained together so as to limit the number of possible errors deduced as possible from the various monitoring statistics to a relatively small number. Software modules track hardware and software configuration changes to the file server, and relating changes in known monitoring statistics to timing of those hardware and software configuration changes. Software modules determine the configuration change most likely to be responsible for a computer assisted diagnosed error, and of suggesting activities to reverse the hardware and software configuration changes so as to place the file server in an operating state.

AUTO-DIAGNOSIS OF PROBLEMS IN AN APPLIANCE OPERATING SYSTEM

THE TECHNICAL APPENDIX TO PATENT APPLICATION ENTITLED

"COMPUTER ASSISTED AUTOMATIC ERROR DETECTION AND DIAGNOSIS OF FILE SERVERS"

INVENTOR: GAURAV BANGA

FILING DATE: DECEMBER 3, 1999

Auto-diagnosis of problems in an appliance operating system

Gaurav Banga gaurav@netapp.com
Network Appliance Inc., 495 East Java Drive, Sunnyvale, CA, 94089

Abstract

The use of network appliances, i.e., computer systems specialized to perform a single function, is becoming increasingly widespread. Network Appliances have many advantages over traditional general-purpose systems such as higher performance/cost metrics, easier configuration and lower costs of management.

Unfortunately, while the complexity of configuration and management of network appliances in normal usage is much lower than that of general-purpose systems, this is not always so in problem situations. The debugging of configuration and performance problems with appliance computers is a task that is similar to the debugging of such problems with general-purpose systems, and requires substantial expertise.

In this paper we examine the issues of *appliance-like* management and performance debugging. We present a number of techniques that we developed to enable *appliance-like* problem diagnosis. We also describe the application of these techniques to a problem auto-diagnosis subsystem that we have built for the Data ONTAP operating system. Our experience with this system indicates a significant reduction in the cost of problem debugging and a much simpler user experience.

1 Introduction

The use of network appliances, i.e., computers specialized to perform a single function, is becoming increasingly widespread. Examples of such appliances are file servers [19, 5], e-mail servers [16, 12], web proxies [20, 4], web accelerators [20, 4, 14] and load balancers [3, 11]. Appliance computers have many potential advantages over traditional general-purpose systems, such as higher performance/cost metrics, simpler configuration and lower costs of management. With the widespread growth in the use of networked systems by the non-expert, mainstream population, all of these advantages have significant importance.

A network appliance is typically constructed using off-the-shelf hardware components. The appliance's service is implemented by custom software running on top of a specialized operating system. (Often the server software is tightly integrated with the core operating system in the same address space.) The appliance's OS is either designed and constructed from scratch, e.g., Network Appliance's Data ONTAP [21], or is a stripped-down version of a general-purpose operating system, e.g., BSDI's Embedded BSD/OS [7].

While network appliances have delivered the promise of higher performance for the same cost vis-a-vis general-purpose systems, the same is not strictly true of their manageability aspects. While the complexity of configuration and management of appliance computers in "normal" circumstances is significantly lower than that of general-purpose systems, the debugging of configuration and performance problems of appliances (when they do occur) remains a task that requires substantial operating system and networking expertise. In this respect, network appliances are somewhat similar to general-purpose systems.

This state of technology is not very surprising: Today, the term "appliance-like" is usually taken to mean "specialized to do a single coherent task well". Specialization of this form has allowed appliance vendors to build and maintain smaller amounts of code than general-purpose computers. The narrower functionality of appliances has enabled simpler configuration, and more aggressive optimizations leading to superior performance. The ability to easily debug configuration and performance problems has been a secondary issue so far, and has not received much attention.

Appliance operating systems often contain significant amounts of code derived from general-purpose operating systems, particularly UNIX. For instance, the BSD TCP/IP protocol code [26] is a common building block in appliance operating systems. Like general-purpose systems, appliance operating systems export a

set of command interfaces that allow users to display values of various statistic counters corresponding to the various events that have occurred during the operation of the system. Some command interfaces display system configuration parameters. As with general-purpose systems, these command interfaces are the key tools to debugging performance and configuration problems with appliance systems.

For example, the TCP/IP code of many appliance systems exports its event statistics and configuration via a variant of the UNIX *netstat* command. When a person debugging a configuration or performance problem suspects a problem in the networking component of the target appliance system, she executes the netstat command (possibly multiple times with its many options) and analyzes the output for aberrations in the counter values from expected "normal" values. Any deviations of these statistics from the norm provide clues to what might be wrong with the system. Using these clues, the person debugging the problem may perform additional observations of the system's statistics, using other commands, and perform with further analysis and corrective actions (such as configuration changes).

The fundamental problem with this style of statistic-inspection based problem diagnosis is the need for human intervention, and specialized networking and performance debugging expertise in the intervening human. For example, consider a workstation that is experiencing poor NFS file access performance. Assume that the cause of this problem is excessive packet loss in the network path between the client and a NFS [22] server due to a Ethernet duplex mismatch at the server. To diagnose this problem today, the person debugging the problem has to isolate the problem to the server, check the packet drop statistics for the transport protocol in use (UDP or TCP) and correlate these statistics with excessive values for CRC errors or late-collisions maintained by the appropriate network interface driver [1]. After this, the problem debugger has to perform additional configuration checks to verify the existence of a duplex mismatch.

For any organization engaged in selling and supporting network appliances, it is very expensive to provide a large number of human experts with this level of expertise for the on-site debugging of customer problems. In the absence of sufficient numbers of human experts, problem FAQs, and semi-interactive troubleshooting guides are commonly used by customers and by the (mostly) non-expert customer support staff of the appliance vendors for diagnosing field problems.

Another limitation of this style of problem debugging is that field problems are usually detected *after* they occur. Problems are first detected by unusual behavior (e.g., poor performance) at the application level and then traced back towards the cause by a human expert through an exhaustive search and pattern-match through the system's statistics, and by the use of further analysis data. While there is usually a well-understood notion of "normal" and "bad" values for the various statistics, there exists no software logic to continuously monitor the statistics, and to catch shifts in their values from "normal" to "bad". Problems (and resulting service outages) which can be avoided by taking timely corrective actions are not avoided.

For all of these reasons, the use of a network appliance can sometimes be a somewhat frustrating experience for a non-expert customer. The subject of this paper is the problem of enabling simple and easy, i.e., *appliance-like*, debugging of the field problems of appliance computers. We describe four techniques, i.e., *continuous statistic monitoring*, *protocol augmentation*, *cross-layer analysis* and *configuration change tracking*, that we have developed to make the diagnosis of appliance problems easier. We also describe the application of these ideas to build an auto-diagnosis system for the Data ONTAP operating system. While our discussion is set in the context of an appliance operating system, most of the ideas that we present are directly applicable to the space of general-purpose operating systems.

The rest of the paper is structured as follows. In the next section, we discuss the nature of common field problems of network appliances. In Section 3, we describe the four techniques that we have developed to diagnose such problems automatically and efficiently. In Section 4, we describe the implementation of the NetApp auto-diagnosis system. Section 5 describes our experience with this auto-diagnosis system. Section 6 covers related work. Section 7, summarizes the paper and offers some directions for future work.

[1]Note that the duplex mismatch cannot be simply avoided as a configuration or installation time automatic check by the server's software; the Ethernet protocol specification does not contain sufficient logic for an end-system to detect a duplex mismatch

2 The nature of field problems with appliance systems

Before getting into the details of what can be done to make the task of debugging appliance performance and configuration problems more simple, it is important to understand the nature of field problems with appliance systems. In this section, we present an overview of the common causes of field problems with appliances and try to give the reader a sense of why it is hard to debug these problems.

For purposes of concrete illustration, the discussion in the remainder of this paper uses the example of a filer server (filer) appliance. A filer provides access to network-attached disk storage to client systems via a variety of distributed file system protocols, such as NFS [22] and CIFS [13]. A useful model is to think of a filer's operating system as two high-performance pipes between a system of disks and a system of network interfaces. One pipe allows for data flow from the disks to the network; the other allows for the reverse flow. For maximum filer performance, it is important for these pipes to be full, i.e. they should have sufficient client load and there should be no bubbles in the pipes.

2.1 Misconfiguration

A leading cause of field problems with network appliances is system misconfiguration. This may seem somewhat paradoxial since by definition an appliance is a simple computer system that has been specially developed to perform a single coherent task. This definition is supposed to allow an appliance system to be simpler to configure and use. In reality, appliances by themselves are usually much simpler than general-purpose systems. However, the task of making appliances work correctly in a real network in a variety of application environments may still have significant configuration complexity.

One major reason for the configuration complexity associated with a network appliance is that an appliance system in use is actually only a part of a potentially complex distributed system. For example, the perceived performance of a filer is the performance of a distributed system consisting of a client system (usually a general-purpose computer system) connected via a potentially complicated network fabric (switches, routers, cables, patch panels etc.) to the filer. These components typically come from different vendors and need to all be configured and functioning correctly for the filer to function at its rated performance. Unfortunately, this does not always happen for a variety of reasons, as discussed below.

First, the client system usually has a fairly complicated and error-prone configuration procedure. The client's configuration complexity is much more so than the filer's because the client is a general-purpose system. Often, the default configurations in which most client systems ship are simply not set for optimal performance. (This issue of default configuration is discussed in somewhat more detail later.) In many cases, the configuration controls are too coarse for any allowable setting to result in good performance for all activities that the general-purpose client may be engaged in.

Second, while most components of the network fabric are appliances (and therefore presumably easier to configure than client systems), there are numerous potential incompatibilities between them. For example, it is not uncommon for implementations of network communication protocols from different vendors to not work with each other. Usually, the corresponding vendor documentation clearly states this incompatibility, but customers try to use the incompatible implementations anyway, and the result is a field problem.

Perhaps more importantly, some commonly used standard network protocols have serious inadequacies. For example, the Ethernet standard includes an "auto-negotiation" protocol for negotiating the link speeds communicating entities. The standard does not allow for reliable negotiation of "duplex" settings. As a result, perfectly legal configuration settings for link and duplex at two communicating endpoints may result in a "duplex-mismatch", a misconfiguration whose effect on a filer's throughput is disastrous.

Furthermore, network components often use protocols that are vendor-specific or are ad-hoc standards. These "early" protocols work well in most situations, but not at all (or poorly) in other circumstances. In the fast moving world of network technology, there is a fair number of ad-hoc, unstandardized, or incomplete protocols in use at any given time. An example of this is the EtherChannel link aggregation protocol. This protocol does not specify the algorithm for performing load balancing of network traffic between the various links of the EtherChannel. Switch vendors have their own propriety methods for this process, often with surprising interactions with how the client systems and the rest of the network elements are set up. These

interactions sometimes have a significant effect on performance and result in field problems.

A second important cause of the configuration complexity associated with a network appliance is the sub-optimal management of configuration parameters. The appliance philosophy is to expose a very small number of configuration parameters at installation. There is a second tier of parameters that are assigned default values which result in good performance in the majority of installations. For some installations with atypical workloads, these settings may not be optimal. There is usually no automatic logic to tune these second tier parameters. In these cases, these knobs may require tuning by an expert for good performance.

With the widespread increase in the variety and number of appliance customers, this "atypical" population can become a significant overall number, potentially resulting in a large number of field problems. This problem of configuration parameter management also exists with general-purpose operating systems, including systems that are used as clients for filers. With general-purpose systems, however, a large number of parameters often need to be tuned for a typical customer environment.

2.2 Capacity problems

A second class of field problems with appliance systems arise because of their poor handling of capacity overloads. Most commonly used general-purpose operating systems, and many appliance operating systems, perform well when the request load to which the system is being subjected lies within the capacity of system, but extremely poorly when the offered load exceeds the capacity of the system [17, 6]. Historically, the problem of poor overload performance of computer systems has been well-known, but deemed of somewhat marginal importance. In most circumstances it is not desirable to operate a system under overload conditions for any length of time; instead, the focus so far has been to avoid overload by trying to ensure that there are always sufficient hardware resources available in order to handle the maximum offered load.

In the filer appliance market, systems are often purchased by customers with a certain client load in mind. The number and types of systems purchased is chosen based on rated capacities of the filers, by in-house benchmarking, or from knowledge based on prior-use of the same type of filer. Filers are however usually assigned rated capacities based on their performance under some standardized benchmark, e.g. the SpecFS (SFS) benchmark [25]. For many customers' sites, the request load profile is significantly different from the SFS profile, and the real capacity of a filer in operation may be very different from its rated capacity. When offered load does exceed "real" capacity, poor performance and a field problem results.

2.3 Hardware and software faults

Last but not least, some field problems with appliances occur because of software and hardware faults. Unlike the other causes of field problems discussed above, faults are the result of some bug in the system's implementation, and usually result in system down-time. For a mature system made by a technically sound organization, the number of field disruptions due to faults should be very small. Appliance systems are perhaps more challenged to achieve this goal than general purpose systems because of the following:

1. Appliance systems often stretch the underlying hardware components closer to their limits than general purpose systems do. Many hardware problems only show up when the hardware is operating close to its rated limits, i.e., only in appliance systems.

2. Appliance systems often use more complicated and heavily optimized software algorithms than general-purpose systems for implementing their specialized functionality. These algorithms are also refined more pro-actively that the algorithms of general-purpose systems with feedback from the field. It is somewhat more challenging to keep the more dynamically changing appliance software stable.

Unlike the other causes for field problems discussed earlier, it is usually relatively easy to trace a problem due to a hardware or software fault to the cause of the problem. Often, the system simply fails to come up or crashes with an error message that describes the fault in question. Field problems due to faults are not discussed further in this paper. The techniques that we describe in this paper to allow for easy debugging of field problems are orthogonal to the problem of reducing the rate of field disruptions due to faults.

2.4 Why are appliance problems hard to debug?

When a field problem occurs with an appliance system due to any of the reasons described above (except faults), it is usually hard to debug. Consider a filer customer who observes performance that is substantially lower than the filer's rated performance. The reason for this performance drop may be a misconfiguration somewhere in the client-to-filer distributed system, i.e., in the client, in the filer, or in the network fabric. Alternately, the problem may be an overloaded filer; this particular environment may have an atypical load and the filer may have a lower capacity for this workload than for the standard SFS workload.

As the end effect of all of these potential causes is usually the same, i.e., poor file access performance as seen from the client system, it is not easy to discern the exact cause of the problem. The problem debugger is forced to perform a sanity check of *all* the components of the client-to-filer distributed system in order to ensure that each component is functioning correctly. For the filer, this implies a verification of all filer subsystems performed by invoking the various statistic commands and analyzing the output for aberrations.

This process is time-consuming, tedious and error-prone. As explained earlier, this task requires a fair amount of expertise. This task is also complicated by the fact that the person debugging the field problem, being a member of the filer vendor's organization, often has no direct access to the system being debugged. In that case, the various statistic commands are executed by the customer who is in communication with the support person via email or phone. This last aspect of the problem debugging process makes it slow, causing large down-time. Combined with the high expectations of "appliance-like" simplicity that most appliance customers have, it makes the problem debugging experience very frustrating for both parties involved, the customer and the support person.

3 Problem auto-diagnosis methods

In this section, we describe a new methodology that we have developed to make the diagnosis of appliance field problems simpler. Our goal in designing this methodology was to enable problem diagnosis to be as automatic, precise and quick as possible. We wanted to eliminate the need for expert human intervention in the problem diagnosis process whenever possible. Furthermore, for those situations where expert manual analysis is necessary, we wanted to provide powerful debugging tools, precise and complete system information and the results of partial auto-analysis to the human expert, allowing for fast diagnosis and smaller down-times.

Our problem diagnosis methodology is based on four specific techniques, i.e., 1) continuous monitoring, 2) protocol augmentation, 3) cross-layer analysis and 4) configuration change tracking. Each of these techniques is described in detail below. In this section, we will focus on the fundamental principles underlying these techniques; the next section will contain specific details about the application of these techniques to an auto-diagnosis subsystem in the Data ONTAP operating system.

We will also briefly discuss issues related to the extendibility of our new problem diagnosis methodology. This feature is important for the problem auto-diagnosis system to be maintainable in the field.

3.1 Continuous monitoring

As described in Section 1, current appliance operating systems maintain a large number of statistics. To help in auto-detecting and diagnosing problems, we have developed a method of continuous statistic analysis layered on top of this statistic collection procedure. Software logic in the appliance system continuously monitors the system for problems, actively analyzing and fixing whatever problems it can. Continuous monitoring has two components to it, a passive part and an active part.

The passive part of continuous monitoring is a statistic monitoring subsystem of the appliance's operating system. This subsystem periodically samples and analyses the statistics being gathered by the operating system. It automatically looks for any aberrant values in these statistics and applies a set of pre-defined rules on any aberrations from expected "normal" values to move the system into one of a a set of error states. For example, a filer may continuously monitor the average response time of NFS requests. A capacity overload situation is flagged when the response time exceeds a high-water mark.

Some abnormal system states may correspond uniquely to specific problems; other states may be indica-

tive of one of a set of possible problems. In the latter case, the continuous monitoring subsystem may also automatically execute specially designed tests in order to pin-point the specific problem with the system. This is the active part of continuous monitoring. For example, a large number of packet losses on a TCP connection at a filer may be indicative of, among other problems, a duplex mismatch at one of the filer's network interfaces or a high level of network congestion in the path from the relevant client to the filer.

Making continuous system monitoring viable involves the following:

- Development of software logic that formally codifies the informal notion of "expected" statistic value. This activity must be performed for all of the statistics that are gathered by the system. The end-result of this activity is a set of equations that test the state of the system and return either "GOOD" or move the system into an "ERROR" state.
- Development of software logic that that selects an appropriate problem pin-pointing procedure when one of several problems is suspected based on observations of aberrant system statistics.
- Development of formal procedures for pin-pointing common field problems of appliances.

Formally codifying the notion of "expected" values of the various statistics is a hard problem. This is because, in general, the normal values of the various system statistics and the relative sets of values that indicate error conditions depend on how a particular system is being used. For example, an average CPU utilization of 70% might be OK for a system that is usually not subject to bursts of load that greatly exceed the average. This value of average CPU utilization may, however, be a big problem for a system whose peak load often exceeds the average by large factors.

To make the development of this logic tractable, it may be necessary to be somewhat conservative in the choice of the specific problems to be characterized. For any particular appliance, this logic can start from being very simple, codifying only the most obvious problems initially, and move towards more complicated checks as the appliance's vendor gain's experience with how the appliance is used in the field. At any point in an appliance's life-cycle, there will be some logic that can be completely automatically executed and its results presented directly to the customer/user. Other, more complicated logic, may attempt to perform partial-analysis and make these results available to a support person looking at the system, should manual debugging be necessary. Still more complicated analysis may be left to the human expert.

The idea behind developing active tests for pin-pointing problems is to try to mimic the activity of problem analysis by a human expert. While debugging a field problem, this person may take a certain set of statistic values as a clue that the system is suffering from one of a certain set of problems. He may then execute a series of carefully constructed tests to verify his hypothesis and pin-point the exact problem. Continuous monitoring with active tests attempts to model this debugging style.

The algorithmic development activity of active tests motivates the next three techniques, i.e., protocol augmentation, cross-layer analysis and state-change monitoring that we describe below. The software logic to trigger these tests is usually straightforward, once the main logic of continuous monitoring is in place.

Of course, continuous monitoring has to be lightweight. It should work with as few system resources as possible and should not impact system performance in any noticeable way. The active component of system monitoring should not affect the system's environment, e.g., the network infrastructure to which it is attached, in any adverse manner. We will discuss some practical aspects related to the user-interface of the continuous monitoring subsystem in the next section.

Once continuous monitoring is in place, it has a large number of benefits. A sizable fraction of the field problems can be auto-diagnosed, without any intervention of the support staff. If expert intervention is needed, all information that is normally gathered by a human expert after (potentially time-consuming) interaction with the customer is already available. Changing system behavior that slowly moves the system into a state of error may be detected early (and corrected) before it results in down-time. An example of this is the auto-detection of increasing average load that is slowly driving a system into capacity overload.

Similarly, other shifts in a system's environment, such as the load mix to which it is subject to, may be auto-detected and suitable action may be initiated. Continuous monitoring may also help an appliance

vendor to tune his product better because he now has access to more detailed information about the various customer environments in which the product operates. In essence, continuous monitoring is like having a dedicated support person attached to every appliance in the installed base, but at a small fraction of the cost.

3.2 Protocol augmentation

The technique of protocol augmentation refers to the process by which a higher-level protocol in a stacked modular system configures and operates a lower-level protocol through a series of carefully chosen configurations and operating loads. The goal of protocol augmentation is to determine the optimal configuration of the lower-level protocol when it is impossible to determine this setting within the protocol itself. This is necessary because the lower-level protocol is either *inadequate*, *incompletely specified* or if one of the communicating entities has a *broken protocol implementation.*

As briefly mentioned in the previous section, some network protocols are *inadequate* in that it is impossible to detect configuration problems of the communicating entities within the protocol itself. An example of this is Ethernet auto-negotiation, which does not always allow for the correct negotiation of the duplex settings of the communicating entities.

Some network protocols are *incompletely specified.* For instance, the algorithms for congestion control were not specified as part of the original TCP protocol standard. Congestion control was incorporated by most TCP implementations much later from a "de-facto" standard published by the researchers who developed these algorithms. Often, such de-facto standards involve areas of the protocol that are not necessary for correctness, and are therefore unenforcable. A TCP implementation that does not perform congestion control correctly may still be able to communicate adequately with other TCP implementations; however, correct congestion control is imperative for system-wide stability and performance.

A number of protocol implementations, especially where unofficial de-facto standards are involved, are *broken.* For example, some commonly used auto-negotiating Gigabit Ethernet devices detect link only if the peer entity is also set to auto-negotiate.

When a problem occurs because of any of the three reasons mentioned above, the continuous monitoring subsystem automatically detects this situation and flags an error condition. If an active test has been developed and associated with the equations that triggered this error state, this active test is then executed. The active test will use protocol augmentation to mimic a human expert in the debugging process. For example, a test designed to detect an Ethernet duplex mismatch may try all legal settings of speed and duplex coupled with initiation of carefully constructed Ethernet traffic. It may analyze the resulting change in system behavior to determine the correct settings for speed and duplex.

Protocol augmentation is a powerful technique that can be used as a guiding framework to formalize many ad-hoc problem debugging techniques used by human experts. Any manual debugging technique that involves a series of steps where system configuration changes alternate with functionality or performance tests is really a form of protocol augmentation. Using this technique as a design guide, we can come up with problem diagnosis procedures that are more precise and systematic than the ad-hoc techniques normally used in manual diagnosis. In the next section, we will describe some examples of the use of this technique in designing automatic problem diagnosis tests for commonly occurring filer problems.

3.3 Cross-layer analysis

Many subsystems of appliance operating systems are implemented as stacked modules. An example of this is the TCP/IP subsystem, which consists of the link layer, the network layer (IP), the transport layer (TCP and UDP) and the application layer organized as a protocol stack. Each layer of a stacked set of modules maintains an independent set of statistics for error conditions and performance metrics. When a problem occurs, it may by manifested as aberrant statistic values in multiple layers in the system. In classical systems, there is no logic that correlates these aberrant statistic values across different system layers.

Cross-layer analysis is a new technique whereby statistic values in different layers of a subsystem are linked together, and co-analyzed. Essentially, we identify paths [18] in subsystems and link together the statistic values in the various layers that each path crosses. When continuous monitoring detects a problem in a path, the various layers of the path can be quickly examined to isolate the specific problem.

As a debugging technique, cross-layer analysis is a formalization of the ad-hoc technique used by human experts in manual problem debugging where an observation of an aberrant statistic value in one layer triggers a study of the statistic values of an adjacent layer. Considering the pipeline analogy of an appliance operating system, cross-layer analysis guides the debugging process by tracing through and ensuring the health of the various layers that implement the disk-to-network pipes. Thinking of an appliance OS as a collection of paths [18], cross-layer analysis is the technique used to isolate a fault in a path.

As a guiding framework, cross-layer analysis can help in the design of logic that causes the continuous monitoring subsystem to trigger the various active tests. For example, the logic to perform a check for a duplex mismatch on a particular network interface may be triggered off by an observation of excessive TCP level packet loss in a connection that goes through this interface. Cross-layer analysis can also guide the design of the statistic data and the statistic collection logic so as to allow problem debugging to be easier. For example, the need to do cross-layer analysis may require a modification of the BSD *tcpstat* and *ipstat* structures so as to keep some statistics on a per flow basis.

3.4 Automatic configuration change tracking

Many field problems with appliance systems are caused by changes in the system's environment. These include system configuration changes and changes in the offered load. As described earlier, there is a lot of value in continuous monitoring of system statistics to notice shifts in metrics like average system load. Likewise, it is useful to track changes in the system's configuration, both explicit as well as implicit. Automatic tracking of system configuration is the fourth technique of our new problem diagnosis methodology.

Automatic tracking of configuration changes is useful in finding the cause of appliance problems that occur after a system has been up and running correctly for some time. This technique also helps in prescribing solutions for the problems found by other auto-diagnosis methods. In many organizations, there are multiple administrators responsible for the IT infrastructure. Configuration change tracking allows for actions of one administrator that result in an appliance problem to be easily reversed by another administrator. This is also useful where administrative boundaries partition the network fabric and the clients from the filer.

The fundamental motivation behind automatic configuration change tracking is to automatically gather information that is asked for by human problem debuggers in a large majority of cases. Anyone familiar with the process of field debugging probably knows that one of the first questions that a customer reporting a problem gets asked by the problem solving expert is: "What has changed recently?" The answer to this is often only loosely accurate (especially in a multi-administrator environment), or even incorrect, depending on the skill-level of the customer/user. Automatic configuration change tracking makes precise and complete state change information available to the problem solver, i.e., the auto-diagnosis logic or a human expert.

Configuration changes are tracked by a special module of the appliance OS. As hinted above, configuration changes are of two types: The first type of changes are explicit, and correspond to state changes initiated by its operator. The second type of changes are implicit, e.g., an event of link-status loss and link-status regain when a cable is pulled out and re-inserted into one of a filer's network interface cards. The system logs all of the explicit and implicit changes. The amount of change information that needs to be kept around is a system design parameter, and may require some experience in getting to optimal for any specific appliance.

Given complete configuration change information, when a problem occurs the various events between the last instance of time which was known to be problem free to the current event are examined and analyzed. The software logic to do this analysis, like the logic for continuous monitoring, is system specific and may need to be evolved over time. In some cases, the auto-diagnosis system can directly infer the cause for the field problem, and report this. In other cases, the set of all applicable configuration changes can be made available to the human debugging the system.

Figure 1 shows the role of the various auto-diagnosis techniques that we have described in the problem diagnosis process. In the figure, dashed lines indicate flow of data while solid lines indicate flow of control. The shaded rectangles indicate stores of data or logic rules. The unshaded rectangles indicate processing steps.

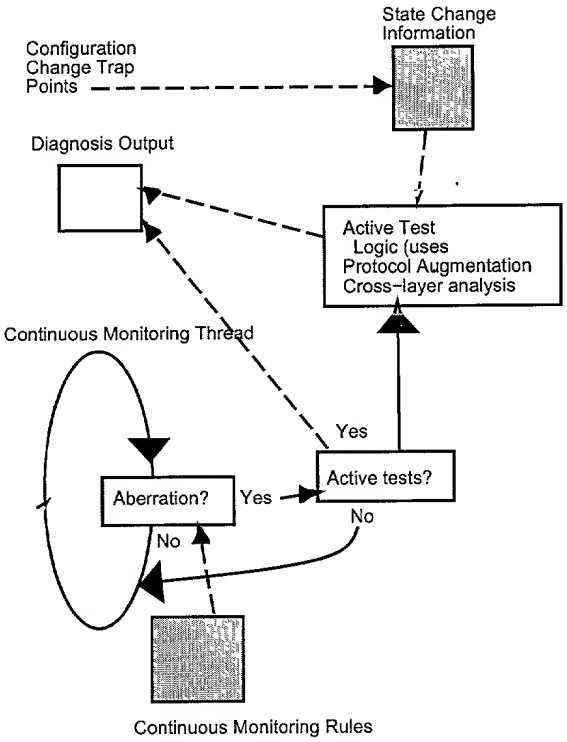


Figure 1: Role of different auto-diagnosis techniques.

3.5 Extendibility issues

It is important for an auto-diagnosis system built around the techniques described above to be extendible. As explained above, the checks and actions performed by the continuous monitoring logic need to be developed in a phased and conservative manner. Each time a new version of this logic is available, a vendor may want to upgrade the systems in the field with this logic, even if the customers do not wish to upgrade the rest of the system. A customer may not wish to take on the risk associated with a new software, or may not want to pay for the release, especially if it does not contain any functionality that the customer needs. It is, however, usually in the vendor's interest to upgrade the auto-diagnosis logic because of the little associated risk and potential benefits of lower support costs.

For example, a problem with an appliance may have been first discovered at a particular customer's installation because of a specific environment change, e.g. the addition of a new model of some hardware in the network fabric. In some cases, significant effort by human experts may be required to debug this problem since it has not been seen before. Ideally, we would like to leverage of this effort by codifying the debugging logic used in this manual diagnosis in the auto-diagnosis logic and upgrading the auto-diagnosis subsystems of *all* the systems in the field. This may save a lot of time and effort by auto-diagnosing subsequent instances of this problem which would otherwise require significant human intervention.

Extendibility can be achieved in a variety of ways. One method is for the continuous monitoring system to have a configuration file containing equations that define the various periodic checks that the continuous monitoring system needs to make and the conditions that trigger movement of the system into an ERROR state, or cause an active subtest to be executed. This requires a language to express the logic of the periodic checks, and an interpreter for this language to be part of the problem auto-diagnosis subsystem.

4 Implementation of the NetApp Auto-diagnosis System

We have implemented a semi-automatic problem diagnosis system (the NetApp Auto-diagnosis System) in the Data ONTAP operating system. This system applies the techniques described in the previous section

to field problems with filers and NetCache appliances. Currently this auto-diagnosis system only targets problems related to the networking portion of Data ONTAP, and some of the interactions of this code with the rest of Data ONTAP. Extension of the auto-diagnosis system to other ONTAP subsystems is in progress.

An interesting social problem that we had to address while developing the auto-diagnosis system was to how not to make the auto-diagnosis logic intrusive. We did not want our expert customers to be turned-off by an "overbearing" problem diagnosis "assistant" and immediately disable the auto-diagnosis system. We also did not want our non-expert customers to be lead off on a side-track by a bug in the auto-diagnosis logic. For this reason, we decided that we would make the auto-diagnosis process semi-automatic initially, and later, as both we and our customers gained experience with the system, make it fully automatic.

In its current form the NetApp Auto-diagnosis System consists of a continuous monitoring subsystem and a set of "diag" (for diagnostic) commands. ONTAP's continuous monitoring logic consists of a thread that wakes up every minute and performs a series of checks on statistics that are maintained by various ONTAP subsystems. These checks may change the state of the system. This logic does not yet perform any output for direct user consumption; nor does this logic execute any active tests. Instead this output is logged internally in ONTAP for consumption by the various "diag" commands.

When the customer or a support person debugging a field problem suspects that the problem lies in the networking portion of ONTAP, she executes the "netdiag" command. The "netdiag" command analyzes the information logged by the continuous monitoring subsystem, performing any active tests that may be called for and reports the results of its analysis, and its recommendations on how to fix the problem to the user. Our plan is to have the computation of the various "diag" commands be performed automatically after the next few releases of ONTAP.

The checks that ONTAP's continuous monitoring system performs have been defined using data from a variety of sources of collected knowledge. These include: 1) FAQs compiled by the NetApp engineering and customer support organizations over the years, 2) troubleshooting guides compiled by NetApp support, 3) historical data from NetApp's customer call record and engineering bug databases, 4) information from advanced ONTAP system administration and troubleshooting courses that are offered to NetApp's customers, and 5) ideas contributed by some problem debugging experts at NetApp.

The list of problems that ONTAP's auto-diagnosis subsystem will address when complete is fairly extensive; due to space considerations we will not cover the complete list. Instead, we will restrict the following discussion to some common networking problems that ONTAP currently attempts to auto-diagnose. At the link layer, ONTAP attempts to diagnose Ethernet duplex and speed mismatches, Gigabit auto-negotiation mismatches, problems due to incorrect setting of store and forward mode on some network interface cards (NICs), link capacity problems, Etherchannel load balancing problems and some common hardware problems. At the IP layer, ONTAP can diagnose common routing problems and problems related to excessive fragmentation. At the transport layer, ONTAP can diagnose common causes of poor TCP performance. At the system level, ONTAP can diagnose problems due to inconsistent information in different configuration files, unavailability or unreachability of important information servers such as DNS and NIS servers, and insufficient system resources for the networking code to function at the load being offered to it.

To see how the techniques described in the previous section are used, consider the link layer diagnosis logic. The continuous monitoring system monitors the different event statistics such as total packets in, total packets out, incoming packets with CRC errors, collisions, late collisions, deferred transmissions etc., that are maintained by the various NIC device drivers. Assume that the continuous monitoring logic notices a large number of CRC errors. Usually, this will also be noticed as poor application-level performance.

Without auto-diagnosis, the manner in which this field problem is handled depends on the skill level and the debugging approach of the person addressing the problem. Some people will simply assume bad hardware and swap the NIC. Other people will first check for a duplex mismatch (if the NIC is an Ethernet NIC) by looking at the duplex settings of the NIC and the corresponding switch port, and if no mismatch is found may try a different cable and a different switch port in succession before swapping the NIC.

With the "netdiag" command, this process is much more formal and precise (Figure 2). The netdiag command first executes a protocol augmentation based test for detecting if there is a duplex mismatch.

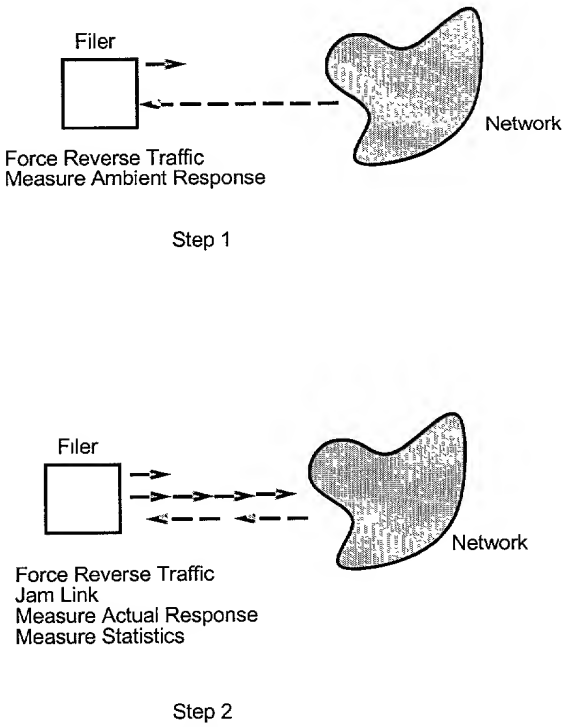


Figure 2: Diagnosing a duplex mismatch using protocol augmentation.

Specifically, the command forces some "reverse traffic" from the other machines on the network to the filer using a variety of different mechanisms in turn until one succeeds. These mechanisms include an ICMP echo-request broadcast, layer 2 echo-request broadcast and TCP/UDP traffic to well-known ports for hosts in the ARP cache of the filer. First the ambient rate of packet arrival at the filer using whatever mechanism that generated sufficient return traffic is measured (Figure 2, Step 1). Next this reverse traffic is initiated again using the same mechanism as before and the outgoing link is jammed with back-to-back packets destined to the filer itself (which will be discarded by the switch). The reverse traffic rate is then measured, along with the number of physical level errors during the jam interval (Figure 2, Step 2). If there is indeed a duplex mismatch, these observations are sufficient to discover it. In this case, the netdiag command prints information on how to fix the mismatch.

If the reason behind the duplex mismatch is a recent change to the filer's configuration parameters, this information will also be inferred by the auto-diagnosis logic and printed for the benefit of the user. If the NIC in question noticed a link-down-up event in the recent past and no CRC errors had been seen before that event, the netdiag command will print out this information as it could indicate a switch port setting change, or a cable change or a switch port change event which might have triggered off the mismatch. This extra information, which is made possible by automatic configuration change tracking, is important because it helps the customer discover the cause of the problem and ensure that it does not repeat. This problem may have been caused by, for example, two administrators inadvertantly acting at cross-purposes.

If there is no duplex mismatch, the netdiag command prints a series of recommendations, such as changing the cable, switch port and the NIC, in the precise order in which they should be tried by the user. The order itself is based on historical data regarding the relative rates of occurrence of these causes.

4.1 Extendibility

Data ONTAP contains an implementation of the Java Virtual Machine. Our approach towards addressing the issue of extendibility is to write most of the auto-diagnosis system in Java. This provides us complete

flexibility to change the auto-diagnosis logic in a new version, ans support older versions of ONTAP. The current version of the auto-diagnosis system is in C; we plan to use Java in the next version of ONTAP.

5 Performance and experience

In this section, we will briefly discuss the performance of the NetApp Auto-diagnosis System, and our experience with how effective it is in making the task of debugging field problems simple.

The continuous monitoring subsystem of ONTAP takes very little resources. Its CPU overhead is less than 0.25% CPU, even on the slowest systems that we ship. The memory footprint is less than 400KB for a typical system. The time that the "netdiag" command takes depends on the configuration of the system and the load on the system. On our slowest system that is configured with the maximum number of allowable network interfaces and is saturated with client load, netdiag takes no more than 15 seconds to execute. On most systems, it takes less than 5 seconds.

The version of ONTAP that contains the NetApp Auto-diagnosis System is still in internal-test. Since it has not yet shipped to our customers, we have not been able to see how well the auto-diagnosis subsystem is able to deal with field problems. Instead, we have been forced to rely on a study in the laboratory in which we simulated a sample of cases from our customer support call record database and measured the effectiveness of the auto-diagnosis system in solving the problems.

We first looked at a sample of 961 calls that came in during the month of September 1999. This set did not include calls that corresponded to hardware or software faults. We also did not consider calls that were related to information needed by the customer about the product. All other types of calls were considered.

Of these 961 calls, 84 had something to do with the networking code and its interactions with the rest of ONTAP. Auto-diagnosis, when simulated on these cases, was able to auto-detect the problem cause for all by 12 of these calls, at a success-rate of 84.5%. The average time that it took the netdiag command to diagnose the problem was approximately 2.5 seconds. We did not even attempt to quantify the secondary effect on the customer's level of satisfaction that auto-diagnosis would cause due the the dramatic reduction in average problem diagnosis time.

Of the 877 calls not corresponding to networking, we performed a static manual analysis in order to figure out which of these problems could be auto-diagnosed by the complete ONTAP auto-diagnosis system. Our study indicates that 634 of these calls (72.3%) could indeed by addressed by some kind of auto-diagnosis. Another 124 (19.6%) of these calls corresponded to problems whose diagnosis could be sped-up significantly by the partial auto-diagnosis information that the diagnosis system provided.

We repeated this simulation and analysis for calls that came in during October 1999. We considered 1023 cases, 97 networking and 926 other. Simulation of the networking cases indicated that auto-diagnosis could solve 88% of these. Static manual analysis of the remaining cases indicated a success-rate of 70%.

In summary, our historical call data seems to indicate that our auto-diagnosis system will be hugely successful in making a lot of problems that currently require human intervention to be automatically addressed. We were unable to directly quantify the increase in simplicity of the problem diagnosis process; the only "relatively weak" metric that we could quantify was turnaround time for the problem, with and without auto-diagnosis. This metric was dramatically lower for auto-diagnosis.

Note to reviewers:

More experience data will be available as the latest version of ONTAP ships. We plan to include this data in the final version of this paper.

6 Related work

To place our work in context, we briefly survey other approaches to field problem diagnosis of networked computer systems, and how our work relates to these techniques.

6.1 Ad-hoc monitoring of UNIX and UNIX-like systems

As briefly described before, most UNIX and UNIX-like operating systems maintain a large number of statistics corresponding to various events that have occurred in the operation of the system. Access to these statistics and other configuration information is provided by a number of command interfaces. Problem

diagnosis usually consists of manually obtaining appropriate statistics and preusing them for aberrant values.

System administrators in some organizations that use a large number of UNIX systems often use a set of home-grown (or commercially available) frameworks of automated scripts to obtain information from a large number of systems and analyse these values. There is a wealth of literature describing these tools [24, 9, 8, 1]. In some ways, this is similar to our technique of continuous monitoring. The information gathered by these automated scripts, however, is at the granularity at which the various operating systems export. This granularity is usually too coarse for complicated auto-diagnosis of the kind that we can perform inside the operating system, with reasonable system overhead. These environments are also limited in the types of active tests that they can perform for pin-pointing problems.

6.2 SNMP

The Simple Network Management Protocol (SNMP) [2] allows for the management of systems in a TCP/IP network within a coherent framework. In the SNMP world, network management consists of *network management stations*, called managers, communicating with the various systems in the network (hosts, routers, terminal servers etc.), called *network elements*. SNMP based management consists of three parts: 1) a Management Information Base (MIB) [15] that defines the various variables (both standardized and vendor-specific) that network elements maintain that can be queried and set by the manager, 2) a set of common structures and an identification schema, called the Structure of Management Information (SMI) [23], that is used to reference the variables in the MIB, and 3) the protocol with which managers and elements communicate, i.e., SNMP.

The system works as follows: The network managers periodically send queries to the elements to get the state of the various elements. Elements send *traps* to managers when certain events happen. The manager may analyse the information available to it via results of queries to build a picture of the health of the network and present this information to the human network manager in a variety of ways. Plugins that extend a managers functionality in a vendor-specific manner are available to handle vendor specific MIBs. An example of a commonly used manager is HP's OpenView [10].

The problem of using SNMP is in some ways similar to the problem of defining appropriate checks for our continuous monitoring system. The various system variables that are checked by continuous monitoring equations correspond to MIB variables. The auto-diagnosis checking logic corresponds to logic in the network manager plugin handling the vendor-specific MIBs. Thus issues that arise in defining the checks that a continuous monitoring system should execute also apply to the design of SNMP logic.

SNMP is different from our system in two main ways: First, SNMP does not really have a parallel for our active tests. A manager can manipulate a network element in some limited fashion, e.g., by using setting appropriate MIB variables. However, this is not nearly as general or as powerful as what can be done by an active test executing in the concerned system itself.

Second, the fact that SNMP depends on the network connectivity to be present between the network elements and the manager limits the types of problems that can be effectively auto-diagnosed by using SNMP. In particular, problems effecting network connectivity may not be easily diagnosed by SNMP.

In some ways the use of SNMP complements our approach. A system of auto-diagnosis using the techniques that we described earlier may be responsible for the "local" health of a system and its interactions with other networking entities that it communicates with. An SNMP based network management infrastructure may provide overall information about the health of a network using information gained by communication with network elements and their auto-diagnosis subsystems.

7 Summary and future work

To summarize, we described some general techniques to enable *appliance-like* debugging of field problems of network appliances. These techniques formalize various ad-hoc debugging techniques that are used in manual debugging of system problems by human experts. These techniques also help in making the task of debugging hard problems manually much simpler and quicker than it currently is.

We have implemented these ideas in the Data ONTAP operating system. Our laboratory studies primed with real historical case data seems to indicate that auto-diagnosis as a methodology is very viable and has

the potential of greatly reducing the complexity of problem analysis that is exposed to the customer.

In terms of future work, we would like to expand our continuous monitoring logic to encompass more complicated problems. As mentioned earlier, we are in the process of making the auto-diagnosis system extendible and easy to re-configure; this problem has a number of interesting issues. It would also be interesting to see a new user-interface paradigm linked with the ideas discussed in this paper that can vary the amount of detail and complexity in the output of the system based on the expertise of the user.

While our discussion has focused on Data ONTAP, from our experience it seems that most of the ideas described in this paper are directly applicable general-purpose operating systems. ONTAP's network code is based on BSD, and much of our auto-diagnosis logic can be directly applied to any BSD based TCP/IP subsystem. We look forward to an application of some of these ideas to general-purpose operating systems.

References

[1] M. Burgess and R. Ralston. Distributed resource administration using cfengine. *Software—Practice and Experience*, 27, 1997.

[2] J. D. Case, M. S. Fedor, M. L. Schoffstall, and C. Davin. Simple Network Management Protocol (SNMP). RFC 1157, May 1990.

[3] Cisco Local Director. http://www.cisco.com/warp/public/cc/cisco/mkt/scale/locald/.

[4] Cobalt CacheRaQ 2. http://www.cobalt.com/products/cache/index.html.

[5] Cobalt NASRaQ. http://www.cobalt.com/products/nas/index.html.

[6] P. Druschel and G. Banga. Lazy Receiver Processing (LRP): A Network Subsystem Architecture for Server Systems. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, Seattle, WA, Oct. 1996.

[7] e/BSD: BSDI Embedded Systems Technology. http://www.BSDI.COM/products/eBSD/.

[8] M. Gomberg, C. Stacey, and J. Sayre. Scalable, Remote Administration of Windows NT. In *Proceedings of the Second Large Installation Systems Administration of Windows NT Conference (LISA-NT)*, Seattle, WA, July 1999.

[9] S. Hansen and T. Atkins. Centralized System Monitoring With Swatch. In *Proceedings of the Seventh Systems Administration Conference (LISA)*, Monterey, CA, Nov. 1993.

[10] HP OpenView. http://www.openview.hp.com/.

[11] IBM SecureWay Network Dispatcher. http://www.ibm.com/software/network/dispatcher/.

[12] Intel InBusiness eMail Station. http://www.intel.com/network/smallbiz/inbusiness_email.htm.

[13] P. J. Leach and D. C. Naik. A Common Internet File System (CIFS/1.0) Protocol. Internet Draft, Network Working Group, Dec. 1997.

[14] J. Liedtke, V. Panteleenko, T. Jaeger, and N. Islam. High-performance caching with the Lava hit-server. In *Proceedings of the USENIX 1998 Annual Technical Conference*, New Orleans, LA, June 1998.

[15] K. McCloghrie and M. T. Rose. Management Information Base for Network management of TCP/IP-based Internets: MIB-II. RFC 1213, Mar. 1991.

[16] Mirapoint Internet Message Server. http://www.mirapoint.com/products/servers/index.asp.

[17] J. C. Mogul and K. K. Ramakrishnan. Eliminating receive livelock in an interrupt-driven kernel. In *Proc. of the 1996 USENIX Technical Conference*, pages 99–111, 1996.

[18] D. Mosberger and L. L. Peterson. Making paths explicit in the scout operating system. In *Proceedings of the Second Symposium on Operating Systems Design and Implementation*, Seattle, WA, Oct. 1996.

[19] Network Appliance – Products – Filers. http://www.netapp.com/products/filer/.

[20] Network Appliance – Products – NetCache. http://www.netapp.com/products/netcache/.

[21] Network Appliance Technical Library. http://www.netapp.com/tech_library/.

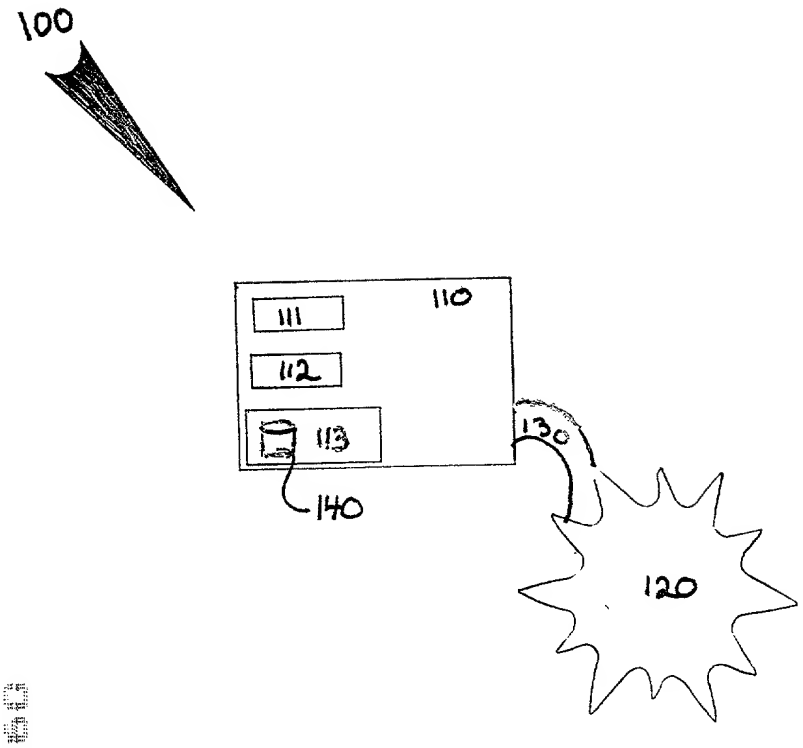
[22] B. Pawlowski, C. Juszczak, P. Staubach, C. Smith, D. Lebel, and D. Hitz. NFS Version 3: Design and Implementation. In *Proceedings of the USENIX 1994 Summer Technical Conference*, Boston, MA, June 1994.

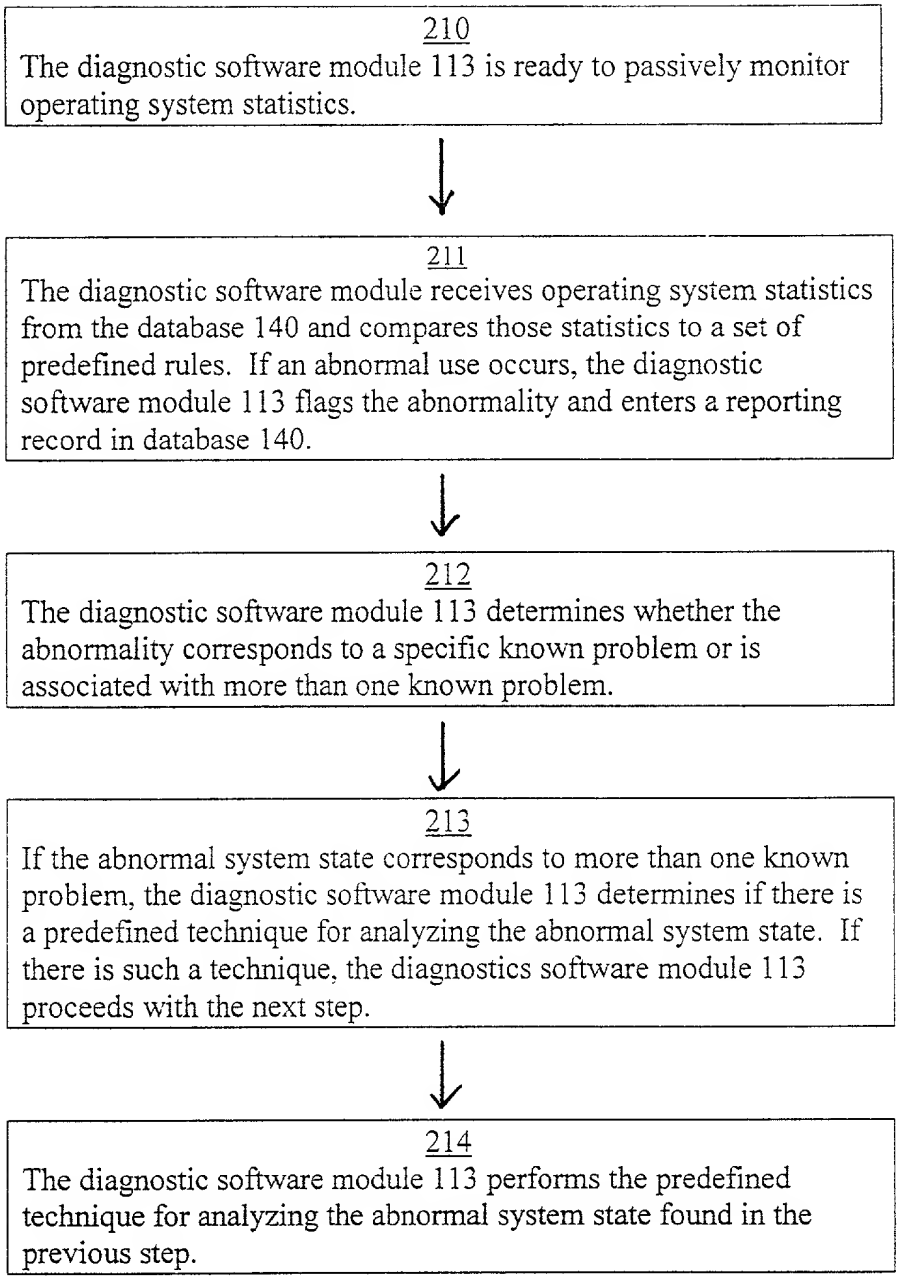
[23] M. T. Rose and K. McCloghrie. Structure and Identification of management Information for TCP/IP-based Internets. RFC 1155, May 1990.

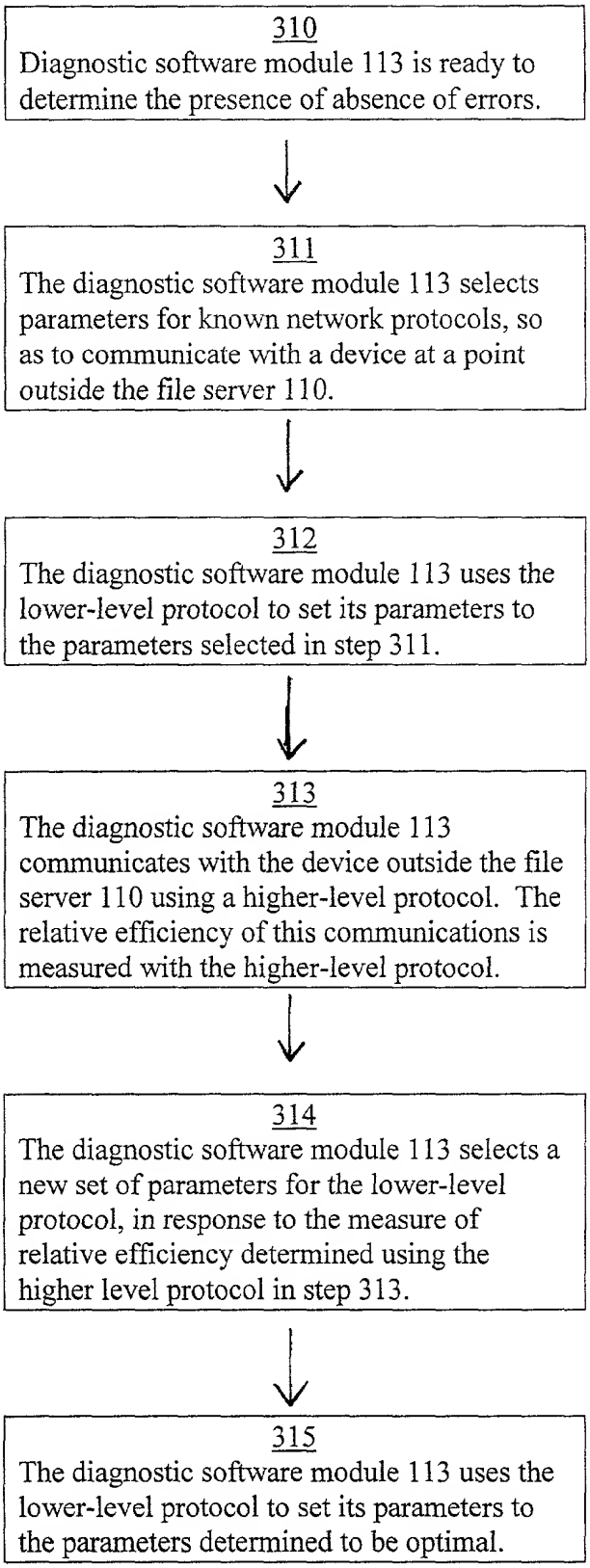
[24] E. Sorenson and S. R. Chalup. RedAlert: A Scalable System for Application Monitoring . In *Proceedings of the Thirteenth Systems Administration Conference (LISA)*, Seattle, WA, Nov. 1999.

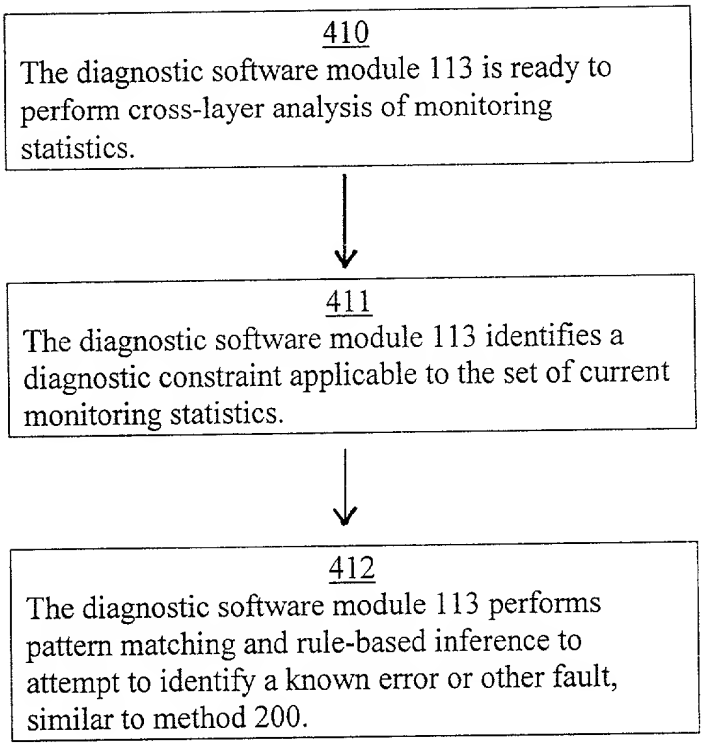
[25] SPEC SFS97. http://www.specbench.org/osg/sfs97/.

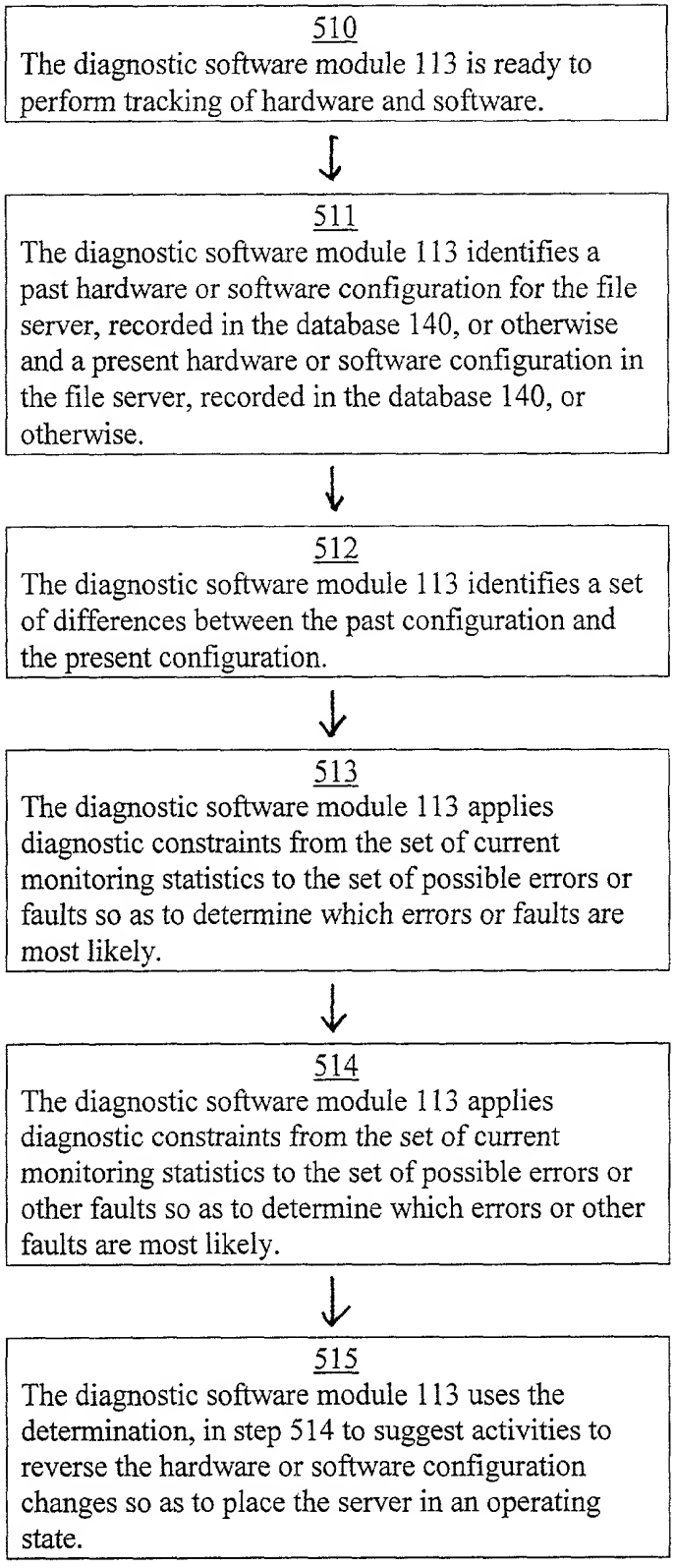
[26] G. R. Wright and W. R. Stevens. *TCP/IP Illustrated Volume 2.* Addison-Wesley, Reading, MA, 1995.


100
110
111
112
113
130
140
120


210
The diagnostic software module 113 is ready to passively monitor operating system statistics.
211
The diagnostic software module receives operating system statistics from the database 140 and compares those statistics to a set of predefined rules. If an abnormal use occurs, the diagnostic software module 113 flags the abnormality and enters a reporting record in database 140.
212
The diagnostic software module 113 determines whether the abnormality corresponds to a specific known problem or is associated with more than one known problem.
213
If the abnormal system state corresponds to more than one known problem, the diagnostic software module 113 determines if there is a predefined technique for analyzing the abnormal system state. If there is such a technique, the diagnostics software module 113 proceeds with the next step.
214
The diagnostic software module 113 performs the predefined technique for analyzing the abnormal system state found in the previous step.


310
Diagnostic software module 113 is ready to determine the presence of absence of errors.
311
The diagnostic software module 113 selects parameters for known network protocols, so as to communicate with a device at a point outside the file server 110.
312
The diagnostic software module 113 uses the lower-level protocol to set its parameters to the parameters selected in step 311.
313
The diagnostic software module 113 communicates with the device outside the file server 110 using a higher-level protocol. The relative efficiency of this communications is measured with the higher-level protocol.
314
The diagnostic software module 113 selects a new set of parameters for the lower-level protocol, in response to the measure of relative efficiency determined using the higher level protocol in step 313.
315
The diagnostic software module 113 uses the lower-level protocol to set its parameters to the parameters determined to be optimal.




510
The diagnostic software module 113 is ready to perform tracking of hardware and software.
511
The diagnostic software module 113 identifies a past hardware or software configuration for the file server, recorded in the database 140, or otherwise and a present hardware or software configuration in the file server, recorded in the database 140, or otherwise.
512
The diagnostic software module 113 identifies a set of differences between the past configuration and the present configuration.
513
The diagnostic software module 113 applies diagnostic constraints from the set of current monitoring statistics to the set of possible errors or faults so as to determine which errors or faults are most likely.
514
The diagnostic software module 113 applies diagnostic constraints from the set of current monitoring statistics to the set of possible errors or other faults so as to determine which errors or other faults are most likely.
515
The diagnostic software module 113 uses the determination, in step 514 to suggest activities to reverse the hardware or software configuration changes so as to place the server in an operating state.